

# PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2004-086921

(43)Date of publication of application : 18.03.2004

(51)Int.Cl.

G06F 15/177

G06F 9/46

(21)Application number : 2003-363920

(71)Applicant : MATSUSHITA ELECTRIC IND CO LTD

(22)Date of filing : 23.10.2003

(72)Inventor : TANAKA TETSUYA  
FUKUDA AKIRA  
TANUMA HITOSHI

(30)Priority

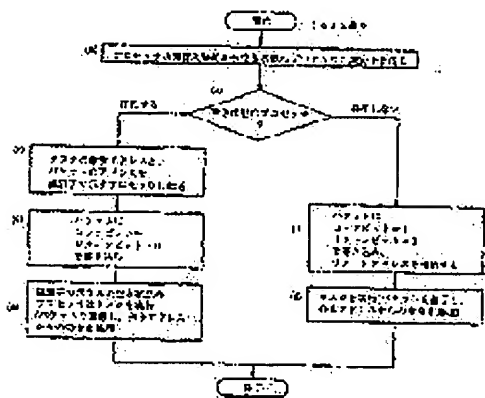
Priority number : 07036836 Priority date : 24.02.1995 Priority country : JP

## (54) MULTIPROCESSOR SYSTEM AND METHOD FOR EXECUTING TASK IN MULTIPROCESSOR SYSTEM THEREOF

(57)Abstract:

PROBLEM TO BE SOLVED: To provide a task executing method suitable to a task of fine granularity.

SOLUTION: This multiprocessor system comprises a step wherein whether or not there is a processor having a 'free state' among the processors 30-32 when a processor which is executing a task T1 among the processors 30-32 generates a new task T2, a step wherein when the processor having the 'free state' is detected, the task T2 begins to be executed by the processor by being assigned to the processor and the state of the processor is changed from the 'free state' to an 'execution state' to store a flag having a first value indicating that the execution of the task T1 is not interrupted. The system also includes a step wherein the execution of the task T1 is interrupted and the task T2 which is interrupted begins to be executed by the processor to store a flag having a second value indicating that the execution of the task T1 has been interrupted when the processor having a 'free state' is not detected.



## LEGAL STATUS

[Date of request for examination]

23.10.2003

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

BEST AVAILABLE COPY

(19) 日本国特許庁(JP)

(12) 公開特許公報(A)

(11) 特許出願公開番号

特開2004-86921

(P2004-86921A)

(43) 公開日 平成16年3月18日(2004.3.18)

(51) Int. Cl.<sup>7</sup>

G06F 15/177

G06F 9/46

F 1

G06F 15/177

6 7 4 A

G06F 9/46

3 6 0 B

テーマコード (参考)

5 B 0 4 5

5 B 0 9 8

審査請求 有 請求項の数 1 O L (全 21 頁)

(21) 出願番号 特願2003-363920 (P2003-363920)  
 (22) 出願日 平成15年10月23日 (2003.10.23)  
 (62) 分割の表示 特願平8-36964の分割  
 原出願日 平成8年2月23日 (1996.2.23)  
 (31) 優先権主張番号 特願平7-36836  
 (32) 優先日 平成7年2月24日 (1995.2.24)  
 (33) 優先権主張国 日本国 (JP)

(71) 出願人 000005821  
 松下電器産業株式会社  
 大阪府門真市大字門真1006番地  
 (74) 代理人 100078282  
 弁理士 山本 秀策  
 (74) 代理人 100062409  
 弁理士 安村 高明  
 (74) 代理人 100107489  
 弁理士 大塩 竹志  
 (72) 発明者 田中 哲也  
 大阪府門真市大字門真1006番地 松下  
 電器産業株式会社内  
 (72) 発明者 福田 晃  
 福岡県三潁郡城島町大字青木島199番地  
 2

最終頁に続く

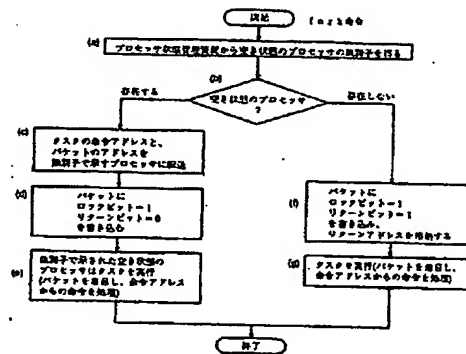
(54) 【発明の名称】 マルチプロセッサシステムおよびマルチプロセッサシステムにおいてタスクを実行する方法

(57) 【要約】

【課題】 細粒度のタスクに適したタスクの実行方法を提供する。

【解決手段】 プロセッサ30～32のうちタスクT1を実行中のプロセッサが新たなタスクT2を生成した場合において、プロセッサ30～32のうち「空き状態」を有するプロセッサがあるか否かを検出するステップと、「空き状態」を有するプロセッサが検出された場合には、タスクT2をそのプロセッサに割り当てることにより、そのプロセッサによるタスクT2の実行を開始し、そのプロセッサの状態を「空き状態」から「実行状態」に変更し、タスクT1の実行が中断されていないことを示す第1の値を有するフラグを格納するステップと、「空き状態」を有するプロセッサが検出されない場合には、タスクT1の実行を中断し、中断したプロセッサによるタスクT2の実行を開始し、タスクT1の実行が中断されたことを示す第2の値を有するフラグを格納するステップとを包含する方法。

【選択図】 図7



## 【特許請求の範囲】

## 【請求項 1】

「空き状態」と「実行状態」とを有する複数のプロセッサを含むマルチプロセッサシステムにおいてタスクを実行する方法であって、

該複数のプロセッサのうち第 1 タスクを実行中の第 1 プロセッサが新たな第 2 タスクを生成した場合において、該第 1 プロセッサが、該第 2 タスクの実行を依頼するために、該複数のプロセッサのうち「空き状態」を有する第 2 プロセッサがあるか否かを検出するステップと、

該検出の結果、「空き状態」を有する第 2 プロセッサが検出された場合には、該第 2 タスクを該第 2 プロセッサに割り当てることにより、該第 2 プロセッサによる該第 2 タスクの実行を開始し、該第 2 プロセッサの状態を「空き状態」から「実行状態」に変更し、該第 1 タスクの実行が中断されていないことを示す第 1 の値を有するフラグを格納するステップと、

該検出の結果、「空き状態」を有する第 2 プロセッサが検出されない場合には、該第 1 プロセッサによる該第 1 タスクの実行を中断し、該第 1 プロセッサによる該第 2 タスクの実行を開始し、該第 1 タスクの実行が中断されたことを示す第 2 の値を有するフラグを格納するステップと

を包含する方法。

## 【発明の詳細な説明】

## 【技術分野】

## 【0001】

本発明は、複数のタスクを並列に実行する複数のプロセッサを含むマルチプロセッサシステムおよびそのマルチプロセッサシステムにおいてタスクを実行する方法に関する。

## 【背景技術】

## 【0002】

近年、マルチプロセッサシステムは汎用計算機の並列処理による高性能化のアプローチの一つとして注目されている。マルチプロセッサシステムにおいては複数のプロセッサを一つのバスに接続し、主記憶装置を共有する共有メモリ型のマルチプロセッサシステムが主に採用されている。

## 【0003】

このようなマルチプロセッサシステムは通常、複数のプロセッサチップをプリント基板上に実装するため、各プロセッサの処理速度に対し、プロセッサ間のバスを用いる通信や同期の処理速度は遅い。そのため、処理単位であるタスクの処理時間がプロセッサ間の通信や同期の時間に対し十分大きい場合に用いられる。この場合のタスクの大きさは中粒度～粗粒度と呼ばれ実行命令数で数 1000 命令程度以上とされている。このように、処理単位を大きくする（粒度を粗くする）ことでタスクの実行時間に対して相対的にプロセッサ通信や同期の時間を小さくしている。

## 【0004】

さらに、近年半導体の集積化技術は急速に発展している。そのため、チップ内に多くの機能ユニットやメモリを搭載することができるようになってきている。マルチプロセッサシステムにおいても今後複数のプロセッサをワンチップに搭載することが可能になると思われる。その場合、プロセッサが接続されるバスもチップ内に入ることになりプロセッサ間の通信や同期の高速化はそこで実行するタスクの粒度の選択肢を広げる。即ち、タスクの大きさが細粒度、命令数で数 10～数 100 命令程度の並列処理が可能になりつつある。今後、このような細粒度のタスクを並列処理することが主流になると予想される。近年注目されているオブジェクト指向プログラミングや関数型言語を用いたプログラミングは、いずれも「細粒度のタスクを並列処理する」ことに合致したものであるからである。

## 【0005】

一方、マルチプロセッサシステムでは、複数のタスクを物理的に限られたプロセッサ数に割り当てることになるため、タスクの実行順序を決定し、どのプロセッサに対しどのタ

タスクを割り当てるかを適切に選択することが行われる。この処理を動的に行うため、まず実行待ちタスクを一次記憶などのタスク管理装置に格納しておき、次に空きプロセッサを検出し、空きプロセッサがある場合は、実行待ちタスクの中から実行すべきタスクを選択し、選択したタスクを空きプロセッサに割り当てるが行われる。このときのタスク選択は仕事全体の実行時間を最小にするなどの目的で行われる。こういったタスクの実行順序を決定し、タスクをどのプロセッサに割り当てるかを決定する処理をスケジューリングといい、決定方法の異なるさまざまなアルゴリズムがある。また、タスク生成によって実行すべきタスクが生じた場合、タスク管理装置に実行待ちタスクとして登録する処理もある。

#### 【0006】

10

図12にマルチプロセッサシステムにおける、従来のプロセッサ割当方法の動作説明図を示す。図12において、プロセッサ2はタスクを生成し、実行待ちのタスクとしてタスク管理装置にタスク4を登録している。プロセッサ0はプロセッサ1が「空き状態」であることを検出すると、タスク管理装置の実行待ちのタスクをスケジューリングアルゴリズムにしたがって一つを選択し、選択されたタスクはプロセッサ0によりプロセッサ1に割り当てられる。このとき、プロセッサ0はスケジューリングの処理を、プロセッサ2はタスク登録の処理をそれぞれ行っている。

#### 【0007】

これは、例えば特開昭63-208948号公報に示すように空きプロセッサ（図12ではプロセッサ1）がタスクレディーキュー（図12ではタスク管理装置）の監視を行い、実行待ちのタスクを自動的に取り出し処理する場合でも、「空き状態」のプロセッサがスケジューリングの処理を行っている。

20

#### 【0008】

また、例えば特開昭62-190548号公報に示されるように、タスクを依頼した依頼プロセッサが、依頼された被依頼プロセッサでのタスクの状態を監視しておき、被依頼プロセッサがタスクの終了を検出した場合、空きプロセッサとなった被依頼プロセッサにほかのタスクを適切に選択し割り当てる方法がある。この方法においては、依頼プロセッサが被依頼プロセッサの状態を監視する処理を行っている。

#### 【0009】

前記したスケジューリング処理やタスクの登録処理、もしくは被依頼プロセッサを監視する処理はそれぞれ内容は異なるもののタスクをプロセッサに割り当て実行するまでのオーバーヘッド即ちタスク処理に付随するオーバーヘッドと考えることができる。図13はタスクの処理時間と前記したオーバーヘッドの処理時間のタイムチャートを示している。図13に示すようにタスクの粒度が中～粗粒度の場合はタスクの処理時間に対してオーバーヘッドの処理時間が相対的に小さいため、オーバーヘッドの処理時間を無視できるレベルにある。

30

#### 【発明の開示】

#### 【発明が解決しようとする課題】

#### 【0010】

しかしながら、上記のようなタスク処理に付随するオーバーヘッドを持つマルチプロセッサシステムにおいて、プロセッサ間の通信や同期を高速化することで細粒度の並列処理を行う場合は、タスクの処理時間に対して相対的にオーバーヘッドの処理時間が大きくなる。

40

#### 【0011】

図14は細粒度の場合のタスクの処理時間とオーバーヘッドの処理時間のタイムチャートを示している。図14に示すようにオーバーヘッドの処理時間はタスクの処理時間に比べて相対的に大きくなり、オーバーヘッドの処理時間が無視できず仕事全体としての処理時間が大きくなるという問題を有する。

#### 【0012】

本発明は上記問題点を鑑み、細粒度の並列処理をプロセッサ間の通信や同期が高速なマルチプロセッサにおいて、タスク管理やスケジューリング、タスク状態の監視を行わないことで、前記したオーバーヘッドをなくし、その代わりのプロセッサに対する動的なタスク

50

割当を一元的、単純かつ高速に行う方法を提供することにある。

【課題を解決するための手段】

【0013】

本発明の方法は、「空き状態」と「実行状態」とを有する複数のプロセッサを含むマルチプロセッサシステムにおいてタスクを実行する方法であって、該複数のプロセッサのうち第1タスクを実行中の第1プロセッサが新たな第2タスクを生成した場合において、該複数のプロセッサのうち「空き状態」を有する第2プロセッサがあるか否かを検出するステップと、「空き状態」を有する第2プロセッサが検出された場合には、該第2タスクを該第2プロセッサに割り当てることにより、該第2プロセッサによる該第2タスクの実行を開始し、該第2プロセッサの状態を「空き状態」から「実行状態」に変更し、該第1タスクの実行が中断されていないことを示す第1の値を有するフラグを格納するステップと、  
10 「空き状態」を有する第2プロセッサが検出されない場合には、該第1プロセッサによる該第1タスクの実行を中断し、該第1プロセッサによる該第2タスクの実行を開始し、該第1タスクの実行が中断されたことを示す第2の値を有するフラグを格納するステップとを包含しており、これにより上記目的が達成される。

【0014】

前記方法は、前記第2タスクの実行が終了した後、前記フラグが前記第1の値と前記第2の値のうちのいずれを有するかを判定するステップと、前記フラグが前記第1の値を有すると判定された場合には、前記第2プロセッサの状態を「実行状態」から「空き状態」に変更するステップと、前記フラグが前記第2の値を有すると判定された場合には、前記  
20 第1タスクの実行が中断されたところから前記第1プロセッサによる前記第1タスクの実行を再開するステップとをさらに包含してもよい。

【0015】

前記複数のプロセッサのそれぞれは、前記複数のプロセッサを互いに識別する識別子を有しており、前記「空き状態」を有する第2プロセッサの検出は、該識別子を用いて行われてもよい。

【0016】

前記複数のプロセッサのそれぞれは、タスクを割り当てる優先順位を決定する優先度を有しており、前記第2プロセッサへの前記第2タスクの割り当ては、該優先度に基づいて行われてもよい。  
30

【0017】

本発明の他の方法は、「空き状態」と「実行状態」とを有する複数のプロセッサを含むマルチプロセッサシステムにおいて、「停止状態」と「第1実行状態」と「第2実行状態」とを有するタスクを実行する方法であって、該複数のプロセッサのうち第1タスクを実行中の第1プロセッサが新たな第2タスクを生成した場合において、該複数のプロセッサのうち「空き状態」を有する第2プロセッサがあるか否かを検出するステップと、「空き状態」を有する第2プロセッサが検出された場合には、該第2タスクを該第2プロセッサに割り当てることにより、該第2プロセッサによる該第2タスクの実行を開始し、該第2プロセッサの状態を「空き状態」から「実行状態」に変更し、該第2タスクの状態を「停止状態」から「第1実行状態」に変更するステップと、「空き状態」を有する第2プロセッサが検出されない場合には、該第1プロセッサによる該第1タスクの実行を中断し、該  
40 第1プロセッサによる該第2タスクの実行を開始し、該第2タスクの状態を「停止状態」から「第2実行状態」に変更するステップとを包含しており、これにより上記目的が達成される。

【0018】

前記方法は、前記第2タスクの実行が終了した後、前記第2タスクの状態を判定するステップと、前記第2タスクが「第1実行状態」を有すると判定された場合には、前記第2プロセッサの状態を「実行状態」から「空き状態」に変更し、前記第2タスクの状態を「第1実行状態」から「停止状態」に変更するステップと、前記第2タスクが「第2実行状態」を有すると判定された場合には、前記第2タスクの状態を「第2実行状態」から「停  
50

止状態」に変更するステップとをさらに包含してもよい。

【0019】

前記複数のプロセッサのそれぞれは、前記複数のプロセッサを互いに識別する識別子を有しており、前記「空き状態」を有する第2プロセッサの検出は、該識別子を用いて行われてもよい。

【0020】

前記複数のプロセッサのそれぞれは、タスクを割り当てる優先順位を決定する優先度を有しており、前記第2プロセッサへの前記第2タスクの割り当ては、該優先度に基づいて行われてもよい。

【0021】

本発明のマルチプロセッサシステムは、複数のタスクを並列に実行する複数のプロセッサと、該複数のプロセッサの状態を管理し、該複数のプロセッサのそれぞれからの問い合わせに応じて「空き状態」のプロセッサの識別子を返す状態管理手段とを備えており、該複数のプロセッサのそれぞれは、新たなタスクが発生した時点で、該状態管理手段に対して「空き状態」のプロセッサがあるか否かを問い合わせる。これにより上記目的が達成される。

【0022】

前記状態管理手段は、該プロセッサからの問い合わせに応答して、現在の状態を次の状態に移させる手段と、該次の状態に基づいて該問い合わせに対する応答を出力する手段とを備えていてもよい。

【0023】

前記マルチプロセッサシステムは、該複数のプロセッサのそれぞれについて、命令キャッシュメモリとデータキャッシュメモリとをさらに備えていてもよい。

【0024】

前記マルチプロセッサシステムは、前記複数のプロセッサ間で命令アドレスおよびパケットアドレスを転送するためのネットワークをさらに備えていてもよい。

【0025】

該複数のタスクのそれぞれは、細粒度であってもよい。

【発明の効果】

【0026】

本発明によれば、あるプロセッサで新たなタスクを生成したときにそのタスクの実行を他あるいは自プロセッサによりただちに開始することができる。このことは、タスクを保持しておく機構やタスクの実行順序をスケジューリングする機構を不要にする。また、実行待ちのタスクを選択し、その選択されたタスクを「空き状態」のプロセッサに割り当てる処理も不要となる。

【0027】

その結果、タスクの処理時間に比較してプロセッサ割り当てに要する時間が少なくてすむ。これにより、マルチプロセッサシステムにおいて、粒度の細かい並列処理の高速化を図ることができる。

【発明を実施するための最良の形態】

【0028】

以下、図面を参照しながら、本発明の実施の形態を説明する。

【0029】

図1は、本発明のマルチプロセッサシステム1の構成を示す。マルチプロセッサシステム1は、集積回路上にインプリメントされる。マルチプロセッサシステム1は、バスを介して主記憶装置2に接続される。

【0030】

マルチプロセッサシステム1は、要素プロセッサユニット10～12を含む。要素プロセッサユニット10～12のそれぞれは、同一の構成を有している。マルチプロセッサシステム1に含まれる要素プロセッサユニットの数は、3に限定されるわけではない。マル

10

20

30

40

50

チプロセッサシステム 1 は、任意の個数の要素プロセッサユニットを含み得る。

【0031】

要素プロセッサユニット 10～12 は、それぞれ、プロセッサ 30～32 と命令キャッシュ (IC) 33～35 とデータキャッシュ (DC) 36～38 とを有している。命令キャッシュ (IC) は、命令を格納するためのキャッシュメモリであり、読み出し専用である。データキャッシュ (DC) は、データを格納するためのキャッシュメモリであり、読み出しと書き込みができる。

【0032】

共有キャッシュ 20 は、要素プロセッサユニット 10～12 によって共有されている。命令セットやデータセットは、通常、主記憶装置 2 に格納されている。データセットは、必要に応じてバスインタフェース 23 を介して共有キャッシュ 20 にロードされる。共有キャッシュ 20 は、主記憶装置 2 と比較して非常に高速に動作することが好ましい。データキャッシュ (DC) と共有キャッシュ 20 とは、アドレスに応じて使い分けられる。例えば、アドレスが 0x00000000～0x7ffffff の範囲内である場合には、データキャッシュ (DC) がアクセスされ、アドレスが 0x80000000～0xffffffff の範囲内である場合には、共有キャッシュ 20 がアクセスされる。

【0033】

要素プロセッサユニット 10～12 は、ネットワーク 21 を介して相互に接続される。ネットワーク 21 は、要素プロセッサユニット 10～12 の相互間で命令アドレスやパケットアドレスを転送するために使用される。ネットワーク 21 は、例えば、3×3 のクロスバースイッチを用いて実現することができる。

【0034】

プロセッサ状態管理装置 22 は、プロセッサ 30～32 の状態を管理する。プロセッサ 30～32 のそれぞれは、「実行状態」および「空き状態」のいずれか一方の状態を有する。

【0035】

プロセッサ 30～32 のそれぞれには固定された優先度が予め割り当てられている。ここでは、プロセッサ 30～33 は、この順番に高い優先度を有していると仮定する。優先度は、複数のプロセッサがプロセッサ状態管理装置 22 を同時にアクセスする場合において、その複数のプロセッサのうちのどのプロセッサにプロセッサ状態管理装置 22 に優先的にアクセスすることを許すかを決定するために使用される。

【0036】

プロセッサ 30～32 のそれぞれは、プロセッサ 30～32 を互いに識別するための識別子 (ID) を有している。典型的には、識別子 (ID) は、番号によって表現される。

【0037】

プロセッサ 30～32 のそれぞれは、その内部にパケットのアドレスを保持する。パケットのアドレスは、例えば、プロセッサ 30～32 の内部のレジスタ (図示せず) に保持される。これにより、プロセッサ 30～32 は、パケットを参照することができる。パケットの詳細は、図 6 を参照して後述される。

【0038】

マルチプロセッサシステム 1 は、複数のタスクを並列に実行する機能を有する。例えば、プロセッサ 30 がタスク T1 を実行しているのと並行して、プロセッサ 31 はタスク T2 を実行することができる。

【0039】

本明細書では、「タスク」とは、命令セットとデータセットとの組であると定義する。命令セットとデータセットとは、いずれも主記憶装置 2 に格納される。プロセッサ 30～32 のそれぞれは、命令セットから命令を逐次読み出し、読み出された命令を解釈実行する。データセットは、プロセッサ 30～32 が命令セットから読み出された命令を解釈実行する際、必要に応じて参照される。また、後述されるパケットは、データセットの少なくとも一部である。

10

20

30

40

50

## 【0040】

図2は、タスクの概念を模式的に示す。この例では、タスク1は、命令セット1とデータセット1の組によって定義され、タスク2は、命令セット1とデータセット2の組によって定義され、タスク3は、命令セット2とデータセット3の組によって定義される。命令セット1～2とデータセット1～3は、それぞれ、主記憶装置2に格納されている。

## 【0041】

図3は、プロセッサ30～32の状態を管理するプロセッサ状態管理装置22の構成例を示す。プロセッサ状態管理装置22は、入力(REQ0～REQ2、RESET0～RESET2)に応答して出力(ID0～ID2、NMP0～NMP2)を提供する組み合わせ回路を含んでいる。その組み合わせ回路は、現在の状態(S)と入力(REQ0～REQ2、RESET0～RESET2)とに応じて次の状態(next S)を決定し、次の状態に対応する出力(ID0～ID2、NMP0～NMP2)を提供する。現在の状態(S)から次の状態(next S)への遷移は、例えば、表1に示される状態遷移表に従って決定される。

## 【0042】



【表 1】

S	REQ	RESET	nextS	ID0	NMP0	ID1	NMP1	ID2	NMP2
001	001	000	011	01	0	--	--	--	--
010	010	000	011	--	--	00	0	--	--
100	100	000	101	--	--	--	--	00	0
011	001	000	111	10	0	--	--	--	--
011	010	000	111	--	--	10	0	--	--
011	011	000	111	10	0	--	1	--	--
101	001	000	111	01	0	--	--	--	--
101	100	000	111	--	--	--	--	01	0
101	101	000	111	01	0	--	--	--	1
110	010	000	111	--	--	00	0	--	--
110	100	000	111	--	--	--	--	00	0
110	110	000	111	--	--	00	0	--	1
111	001	000	111	--	1	--	--	--	--
111	010	000	111	--	--	--	1	--	--
111	100	000	111	--	--	--	--	--	1
111	011	000	111	--	1	--	1	--	--
111	101	000	111	--	1	--	--	--	1
111	110	000	111	--	--	--	1	--	1
111	111	000	111	--	1	--	1	--	1
011	000	001	010	--	--	--	--	--	--
011	000	010	001	--	--	--	--	--	--
011	001	010	101	10	0	--	--	--	--
011	010	001	110	--	--	10	0	--	--
101	000	001	100	--	--	--	--	--	--
101	000	100	001	--	--	--	--	--	--
101	001	100	011	01	0	--	--	--	--
101	100	001	110	--	--	--	--	01	0
110	000	010	100	--	--	--	--	--	--
110	000	100	010	--	--	--	--	--	--
110	010	100	011	--	--	00	0	--	--
110	100	010	101	--	--	--	--	00	0
111	000	001	110	--	--	--	--	--	--
111	000	010	101	--	--	--	--	--	--
111	000	100	011	--	--	--	--	--	--
111	000	101	100	--	--	--	--	--	--
111	000	110	010	--	--	--	--	--	--
111	001	010	001	--	1	--	--	--	--
111	001	100	011	--	1	--	--	--	--
111	001	110	001	--	1	--	--	--	--
111	010	001	110	--	--	--	1	--	--
111	010	100	011	--	--	--	1	--	--
111	010	101	010	--	--	--	1	--	--
111	100	001	110	--	--	--	--	--	1
111	100	010	101	--	--	--	--	--	1
111	100	011	100	--	--	--	--	--	1
111	011	100	011	--	1	--	1	--	--
111	101	010	101	--	1	--	--	--	--
111	110	001	110	--	--	--	1	--	1

図 3 において、S は現在の状態、Next S は次の状態を示す。これらの状態は、プロセッサ 30～32 の状態を示す。例えば、S = 001 は、プロセッサ 30 の状態が「実行状態」であり、プロセッサ 31 とプロセッサ 32 の状態が「空き状態」であることを示している。Next S についても同様である。

## 【0043】

図 3 において、REQ0～REQ2 は、プロセッサ 30～32 からプロセッサ状態管理装置 22 に入力されるリクエストを表す。これらのリクエストは、「空き状態」のプロセッサの識別子を得ることをプロセッサ状態管理装置 22 に依頼するものである。表 1 では、REQ0～REQ2 をまとめて REQ と表記している。例えば、REQ = 101 は、REQ0 が 1（アサート）であり、REQ1 が 0（ネゲート）であり、REQ2 が 1（アサート）であることを示している。

10

20

30

50

## 【0044】

図3において、RESET0～RESET2は、プロセッサ30～32からプロセッサ状態管理装置22に入力されるリセットを表す。これらのリセットは、プロセッサ状態管理装置22内に保持されているプロセッサ30～32の状態を「実行状態」から「空き状態」に変更することをプロセッサ状態管理装置22に依頼するものである。表1では、RESET0～RESET2をまとめてRESETと表記している。例えば、RESET=010は、RESET0が0（ネゲート）であり、RESET1が1（アサート）であり、RESET2が0（ネゲート）であることを示している。

## 【0045】

図3において、ID0～ID2は、プロセッサ30～32からのリクエストに対して「10  
空き状態」のプロセッサの識別子を通知する信号を表す。これらの信号は、プロセッサ30～32からのリクエストに応答してプロセッサ状態管理装置22から出力される。ID0～ID2の値の意味は、以下のとおりである。

## 【0046】

00：プロセッサ30が「空き状態」である。

## 【0047】

01：プロセッサ31が「空き状態」である。

## 【0048】

10：プロセッサ32が「空き状態」である。

## 【0049】

図3において、NMP0～NMP2は、プロセッサ30～32からのリクエストに対し  
て「空き状態のプロセッサが存在しない」旨を通知する信号を表す。これらの信号は、プロセッサ30～32からのリクエストに  
応答してプロセッサ状態管理装置22から出力される。NMP0～NMP2の値の意味は、以下のとおりである。

## 【0050】

0：「空き状態」のプロセッサが存在する。「空き状態」のプロセッサの識別子は、ID0～ID2の値によって示される。

## 【0051】

1：「空き状態」のプロセッサが存在しない。この場合、ID0～ID2の値は、don't careである。

## 【0052】

以下、図4と図5とを参照して、プロセッサ状態管理装置22の機能および動作を説明する。プロセッサ状態管理装置22は、マルチプロセッサシステムに含まれるすべてのプロセッサの状態を管理する。具体的には、プロセッサ状態管理装置22は、プロセッサの識別子とプロセッサの状態とを一对一にしてプロセッサ状態管理装置22内に保持する。プロセッサの識別子は、複数のプロセッサを互いに識別するために使用される。典型的には、プロセッサの識別子は整数で表現される。プロセッサの状態は、「実行状態」か「空き状態」かのいずれかである。

## 【0053】

プロセッサ状態管理装置22は、あるプロセッサからのリクエストに  
応答して、「空き状態」のプロセッサが存在するか否かを判定する。「空き状態」のプロセッサが存在した場合には、プロセッサ状態管理装置22は、その「空き状態」のプロセッサの識別子をそのリクエストを発したプロセッサに返す。「空き状態」のプロセッサが存在しなかった場合には、プロセッサ状態管理装置22は、「空き状態のプロセッサが存在しない」旨のメッセージをそのリクエストを発したプロセッサに返す。

## 【0054】

「空き状態」のプロセッサが複数個存在する場合には、プロセッサ状態管理装置22は、「空き状態」の複数のプロセッサのうち優先度の最も高いプロセッサの識別子をそのリクエストを発したプロセッサに返す。また、複数のプロセッサからのリクエストが同時にプロセッサ状態管理装置22に到達した場合には、そのリクエストを発した複数のプロセ

ッサのうち優先度の高いものから順に上述した処理が行われる。

【0055】

図4(a)および(b)は、プロセッサ状態管理装置22の動作の一例を示す。プロセッサ状態管理装置22は、4つのプロセッサ0~3の状態を管理している。図4(a)に示す例では、プロセッサ0とプロセッサ1の状態は「実行状態」であり、プロセッサ2とプロセッサ3の状態は「空き状態」である。プロセッサ0からのリクエストとプロセッサ1からのリクエストがプロセッサ状態管理装置22に入力される。

【0056】

プロセッサ状態管理装置22は、プロセッサ0からのリクエストに回答して、「空き状態」のプロセッサ2の識別子をプロセッサ0に返し、プロセッサ1からのリクエストに回答して、「空き状態」のプロセッサ3の識別子をプロセッサ1に返す(図4(b)参照)。 「空き状態」のプロセッサの識別子は、プロセッサの優先度に従って返される。また、プロセッサ状態管理装置22は、プロセッサ状態管理装置22内に保持されているプロセッサ2の状態を「空き状態」から「実行状態」に変更し、プロセッサ3の状態を「空き状態」から「実行状態」に変更する。

【0057】

図5(a)および(b)は、プロセッサ状態管理装置22の動作の他の一例を示す。プロセッサ状態管理装置22は、4つのプロセッサ0~3の状態を管理している。図5(a)に示す例では、プロセッサ0とプロセッサ1とプロセッサ2の状態は「実行状態」であり、プロセッサ3の状態は「空き状態」である。プロセッサ0からのリクエストとプロセッサ1からのリクエストがプロセッサ状態管理装置22に入力される。

【0058】

プロセッサ状態管理装置22は、プロセッサ0からのリクエストに回答して、「空き状態」のプロセッサ3の識別子をプロセッサ0に返し、プロセッサ1からのリクエストに回答して、「空き状態のプロセッサが存在しない」旨のメッセージをプロセッサ1に返す(図5(b)参照)。「空き状態のプロセッサが存在しない」旨のメッセージは、例えば、プロセッサ状態管理装置22から出力されるリターンコードの値によって表される。「空き状態」のプロセッサの識別子は、プロセッサの優先度に従って返される。また、プロセッサ状態管理装置22は、プロセッサ状態管理装置22内に保持されているプロセッサ3の状態を「空き状態」から「実行状態」に変更する。

【0059】

図4と図5に示される例では、プロセッサ状態管理装置22によって管理されるプロセッサの数は4である。しかし、これは、説明の便宜上のためであり、本発明が4つのプロセッサを有するマルチプロセッサシステムに限定されるわけではない。本発明は、任意の数のプロセッサを含むマルチプロセッサシステムに適用され得る。

【0060】

図6は、パケット50の構成を示す。パケット50は、ロックビットを格納するロックビット領域51と、リターンビットを格納するためのリターンビット領域52と、リターンアドレスを格納するためのリターンアドレス領域53と、引数を格納するための引数領域54と、戻り値を格納するための戻り値領域55とを有している。パケット50は、タスク毎に共有メモリ20上に確保され、タスクに所有される。これ以降、「タスクに所有されたパケット」を単に「タスクのパケット」と呼ぶ。パケット50は、タスク間のデータの受け渡しやタスクの情報を保持するために使用される。

【0061】

パケット50のロックビット領域51には、ロックビットが格納される。ロックビットは、パケット50を所有するタスクが実行中である間、他のタスクからその実行中のタスクへのアクセスを禁止するか否かを示す。ロックビットが"1"であることは、アクセスを禁止していることを示す。ロックビットが"0"であることは、アクセスを禁止していないことを示す。

【0062】

パケット50のリターンビット領域52には、リターンビットが格納される。リターンビットは、パケット50を所有するタスクを実行する前に、他のタスクを中断したか否かを示す。リターンビットが"0"であることは、「パケット50を所有するタスクを実行する前に、他のタスクを中断していない」ことを示す。これは、「空き状態」のプロセッサにパケット50を所有するタスクが割り当てられた場合に相当する。リターンビットが"1"であることは、「パケット50を所有するタスクを実行する前に、他のタスクを中断した」ことを示す。これは、「空き状態」のプロセッサが存在しなかったため、タスクを実行中のプロセッサがそのタスクの実行を中断して、パケット50を所有する別のタスクを実行する場合に相当する。

10

## 【0063】

パケット50のリターンアドレス領域53には、リターンアドレスが格納される。リターンアドレスは、リターンビットが"1"である場合にのみ参照される。リターンアドレスは、中断されたタスクへの戻りアドレスを示す。

## 【0064】

パケット50の引数領域54には、パケット50を所有するタスクへの引数が格納される。

## 【0065】

パケット50の戻り値領域55には、パケット50を所有するタスクの実行結果である戻り値が格納される。

20

## 【0066】

図7は、プロセッサ30～32がfork命令を解釈実行する手順を示す。プロセッサ30～32は、主記憶装置2に格納されている命令セットから命令を読み出す。読み出された命令がfork命令である場合には、プロセッサ30～32は、図7に示す処理を実行する。

## 【0067】

以下、図7を参照して、プロセッサ30がfork命令を解釈実行する手順をステップごとに詳細に説明する。他のプロセッサ31および32がfork命令を解釈実行する場合も同様である。なお、fork命令は、オペランドとして、新たなタスクの処理内容を示す命令列の先頭アドレス（以降、単に命令アドレスという）と新たなタスクのパケット50のアドレス（以降、単にパケットアドレスという）をとる。

30

## 【0068】

ステップ(a)：プロセッサ30は、「空き状態」のプロセッサが存在するか否かをプロセッサ状態管理装置22に問い合わせる。このような問い合わせは、例えば、プロセッサ30がプロセッサ状態管理装置22にリクエスト（REQ0=1）を送ることにより達成される。プロセッサ状態管理装置22は、そのリクエストに応答して「空き状態」のプロセッサが存在するか否かを判定する。

## 【0069】

「空き状態」のプロセッサが存在する場合には、プロセッサ状態管理装置22は、その「空き状態」のプロセッサの識別子をプロセッサ30に返す。「空き状態」のプロセッサの識別子は、例えば、プロセッサ30がプロセッサ状態管理装置22から出力されるID0の値を参照することによって得られる。「空き状態」のプロセッサが複数個存在する場合には、優先度の最も高いプロセッサの識別子が得られる。また、複数のプロセッサが同時にfork命令を解釈実行する場合には、優先度の高いプロセッサから順にfork命令を解釈実行する。このようにして、プロセッサ30は、「空き状態」のプロセッサの識別子を取得する。

40

## 【0070】

「空き状態」のプロセッサが存在しない場合には、プロセッサ状態管理装置22は、「空き状態のプロセッサが存在しない」旨のメッセージをプロセッサ30に返す。「空き状態のプロセッサが存在しない」旨のメッセージは、例えば、プロセッサ30がプロセッサ

50

状態管理装置 22 から出力される NMP 0 の値を参照することによって得られる。

【0071】

ステップ (b) : 「空き状態」のプロセッサが存在した場合には、プロセッサ 30 は、ステップ (c) ~ (e) の処理を行う。「空き状態」のプロセッサが存在しない場合には、プロセッサ 30 は、ステップ (f) ~ (g) の処理を行う。

【0072】

ステップ (c) : ここでは、「空き状態」のプロセッサは、プロセッサ 31 であると仮定する。この場合、プロセッサ 30 は、fork 命令のオペランドとして与えられたタスクの命令アドレスとタスクのパケットアドレスとをネットワーク 21 を介してプロセッサ 31 に転送する。

10

【0073】

ステップ (d) : プロセッサ 30 は、fork 命令のオペランドとして与えられたタスクのパケットアドレスによって指定されるパケット 50 のロックビット領域 51 に "1" を書き込み、リターンビット領域 52 に "0" を書き込む。その後、プロセッサ 30 は、fork 命令の処理を完了し、次の命令の処理を行う。

【0074】

ステップ (e) : プロセッサ 31 は、ネットワーク 21 を介してプロセッサ 30 からタスクの命令アドレスとタスクのパケットアドレスとを受け取る。プロセッサ 31 は、受け取ったパケットアドレスによって指定されるパケット 50 を参照しながら、受け取った命令アドレスによって指定される命令から処理を開始する。

20

【0075】

以上のステップ (a) ~ (e) により、プロセッサ 30 は、プロセッサ 31 によって実行される処理とは異なる処理を独立に実行することとなる。すなわち、プロセッサ 30 とプロセッサ 31 とによって並列処理が開始される。fork 命令の処理はここで終了する。

【0076】

ステップ (f) : プロセッサ 30 は、fork 命令のオペランドとして与えられたタスクのパケットアドレスによって指定されるパケット 50 のロックビット領域 51 に "1" を書き込み、リターンビット領域 52 に "1" を書き込む。また、fork 命令の次の命令のアドレスをリターンアドレス領域 53 に書き込む。プロセッサ 30 は、実行中のタスクを中断する。

30

【0077】

ステップ (g) : プロセッサ 30 は、fork 命令のオペランドとして与えられたタスクのパケットアドレスによって指定されるパケット 50 を参照しながら、fork 命令のオペランドとして与えられたタスクの命令アドレスによって指定される命令から処理を開始する。fork 命令の処理はここで終了する。

【0078】

以下、図 8 を参照して、プロセッサ 30 が unlock 命令を解釈実行する手順をステップごとに詳細に説明する。他のプロセッサ 31 および 32 が unlock 命令を解釈実行する場合も同様である。

40

【0079】

ステップ (h) : プロセッサ 30 は、実行中のタスクが所有するパケット 50 のリターンビット領域 52 の値が "0" であるか否かを判定する。リターンビット領域 52 の値が "0" であることは、プロセッサ 30 が処理を中断したタスクが存在しないことを示す。従って、リターンビット領域 52 の値が "0" である場合には、プロセッサ 30 は、ステップ (i) の処理を行う。リターンビット領域 52 の値が "1" であることは、プロセッサ 30 が処理を中断したタスクが存在することを示す。従って、リターンビット領域 52 の値が "1" である場合には、プロセッサ 30 は、ステップ (j) の処理を行う。

【0080】

ステップ (i) : プロセッサ 30 は、実行中のタスクが所有するパケット 50 のロック

50

ビット領域51に"0"を書き込み、プロセッサ30の状態を「空き状態」にする。「空き状態」となったプロセッサ30は、これ以降の処理を行わない。unlock命令の処理はここで終了する。

【0081】

ステップ(j)：プロセッサ30は、実行中のタスクが所有するパケット50のロックビット領域51に"0"を書き込む。さらに、プロセッサ30は、リターンアドレス領域53に格納されているアドレスからの命令を処理することにより、中断されたタスクを復帰させる。unlock命令の処理はここで終了する。

【0082】

表2は、fork命令およびunlock命令の解釈実行に応答して、マルチプロセッサシステムの状態がどのように遷移するかを示す。表2に示される例では、マルチプロセッサシステムは、プロセッサP1とプロセッサP2とを有していると仮定する。

【0083】

【表2】

イベント	現在の状態				次の状態			
	P1	P2	T1	T2	P1	P2	T1	T2
P1.fork	RUN T1	IDLE	EX1	STOP	RUN T1	RUN T2	EX1	EX1
P2.unlock	RUN T1	RUN T2	EX1	EX1	RUN T1	IDLE	EX1	STOP
P1.fork	RUN T1	RUN other	EX1	STOP	RUN T2	RUN other	EX1	EX2
P1.unlock	RUN T2	RUN other	EX1	EX2	RUN T1	RUN other	EX1	STOP

図9に示されるように、マルチプロセッサシステムの状態は、プロセッサの状態とタスクの状態とに区分される。

【0084】

プロセッサは、2つの状態を有する。一方の状態は「空き状態 (IDLE)」であり、他方の状態は「実行状態 (RUN)」である。これらの状態は、プロセッサ状態管理装置22によって管理されている状態と同じものである。プロセッサの状態が「実行状態 (RUN)」である場合には、そのプロセッサはいずれかのタスクを実行中である。

【0085】

タスクは、3つの状態を有する。1つ目の状態は「停止状態 (STOP)」であり、2つ目の状態は「第1実行状態 (EX1)」であり、3つ目の状態は「第2実行状態 (EX2)」である。「停止状態 (STOP)」は、プロセッサがタスクの実行を待っている状態であるかタスクの実行を終了した状態である。「第1実行状態 (EX1)」は、他のタスクの実行を中断することなく現在のタスクが実行されている状態である。「第2実行状態 (EX2)」は、他のタスクの実行を中断してその後現在のタスクが実行されている状態である。プロセッサの状態が「実行状態 (RUN)」である場合には、そのプロセッサに実行されているタスクの状態は、「第1実行状態 (EX1)」と「第2実行状態 (EX2)」のうちのいずれかである。

【0086】

表2を再び参照して、マルチプロセッサシステムの状態がどのように遷移するかを説明する。マルチプロセッサシステムの状態は、イベントの発生に응答して、そのイベントと現在の状態に基づいて次の状態に遷移する。ここで、「Px.fork」という表記は、

「プロセッサP<sub>x</sub>がfork命令を実行した」というイベントが発生したことを表し、「P<sub>x</sub>.unlock」という表記は、「プロセッサP<sub>x</sub>がunlock命令を実行した」というイベントが発生したことを表す。

【0087】

表2の第1行は、プロセッサP<sub>1</sub>が「実行状態」（タスクT<sub>1</sub>を実行中）であり、プロセッサP<sub>2</sub>が「空き状態」であり、タスクT<sub>1</sub>が「第1実行状態」であり、タスクT<sub>2</sub>が「停止状態」である場合において、「プロセッサP<sub>1</sub>がfork命令を実行した」というイベントに応答して、プロセッサP<sub>2</sub>の状態が「空き状態」から「実行状態」（タスクT<sub>2</sub>を実行中）に変更され、タスクT<sub>2</sub>の状態が「停止状態」から「第1実行状態」に変更されることを示す。このように状態が遷移するのは、新たなタスクT<sub>2</sub>が生成された時点でタスクT<sub>2</sub>が「空き状態」のプロセッサP<sub>2</sub>に割り当てられるからである。 10

【0088】

表2の第2行は、表2の第1行における次の状態が現在の状態である場合において、「プロセッサP<sub>2</sub>がunlock命令を実行した」というイベントに応答して、プロセッサP<sub>2</sub>の状態が「実行状態」（タスクT<sub>2</sub>を実行中）から「空き状態」に変更され、タスクT<sub>2</sub>の状態が「第1実行状態」から「停止状態」に変更されることを示す。

【0089】

表2の第3行は、プロセッサP<sub>1</sub>が「実行状態」（タスクT<sub>1</sub>を実行中）であり、プロセッサP<sub>2</sub>が「実行状態」（他のタスクを実行中）であり、タスクT<sub>1</sub>が「第1実行状態」であり、タスクT<sub>2</sub>が「停止状態」である場合において、「プロセッサP<sub>1</sub>がfork命令を実行した」というイベントに応答して、プロセッサP<sub>1</sub>の状態が「実行状態」（タスクT<sub>1</sub>を実行中）から「実行状態」（タスクT<sub>2</sub>を実行中）に変更され、タスクT<sub>2</sub>の状態が「停止状態」から「第2実行状態」に変更されることを示す。このように状態が遷移するのは、新たなタスクT<sub>2</sub>が生成された時点で「空き状態」のプロセッサが存在しないため、プロセッサP<sub>1</sub>がタスクT<sub>1</sub>の実行を中断してタスクT<sub>2</sub>の実行を開始するからである。 20

【0090】

表2の第4行は、表2の第3行における次の状態が現在の状態である場合において、「プロセッサP<sub>1</sub>がunlock命令を実行した」というイベントに応答して、プロセッサP<sub>1</sub>の状態が「実行状態」（タスクT<sub>2</sub>を実行中）から「実行状態」（タスクT<sub>1</sub>を実行中）に変更され、タスクT<sub>2</sub>の状態が「第2実行状態」から「停止状態」に変更されることを示す。 30

【0091】

以下、fork命令とunlock命令を含むプログラムを並列処理する場合におけるマルチプロセッサシステム1の動作を説明する。

【0092】

図10は、1から4までの和（1+2+3+4）を二分木に基づいて計算するプログラムの手順を示す。このプログラムは、mainとsumの2つの部分に分かれており、mainは主プログラム、sumは再帰呼び出し可能でかつ並列処理可能なサブルーチンである。sumはnとmの2つの引数を取り、n+1からmまでの和を求めるものである。mainはn=0、m=4を引数としてsumを呼び出すものである。 40

【0093】

まず、初期状態として、プロセッサ30はmainを実行していると仮定する。プロセッサ30の状態は「実行状態」である。また、プロセッサ31およびプロセッサ32の状態は「空き状態」とすると仮定する。

【0094】

以下、プログラムの各ステップ（A）～（H）について、マルチプロセッサシステム1がどのように動作するかを詳細に説明する。

【0095】

ステップ（A）：プロセッサ30は、n=0、m=4を引数としてsumサブルーチン 50

を実行する。具体的には、プロセッサ30は、共有キャッシュメモリ20上にパケット50 (Pk1) を確保し、そのパケット50 (Pk1) の引数領域54に値0と値4とを格納する。次に、プロセッサ30は、sumの命令の先頭アドレスとパケット50 (Pk1) の先頭アドレスとをオペランドとして、exec命令を実行する。exec命令とは、図7に示すfork命令の処理手順のうちステップ(f)と(g)のみに対応する命令である。exec命令は、fork命令と同様にして、オペランドとしてタスクの命令アドレスとタスクのパケットアドレスをとる。

【0096】

プロセッサ30は、パケット50 (Pk1) のロックビット領域51に"1"を書き込み、パケット50 (Pk1) のリターンビット領域52に"1"を書き込み、リターンアドレス領域53にexec命令の次の命令のアドレスを格納する(図7のステップ(f)を参照)。また、プロセッサ30は、パケット50 (Pk1) を参照しながらsumの命令の実行を開始する(図7のステップ(g)を参照)。

【0097】

ステップ(B)：プロセッサ30は、パケット50 (Pk1) から引数nと引数mを読み出し、(n+1)とmとを比較する。(n+1)とmが等しい場合には、処理はステップ(G)に進み、その他の場合には、処理はステップ(C)に進む。sumサブルーチンがmainから最初に呼ばれた場合には、n=0、m=4であるから、(n+1)とmとは等しくない。従って、処理は、ステップ(C)に進む。

【0098】

ステップ(C)：プロセッサ30は、 $k = (n+m) \div 2$ を計算する。ここで、 $(n+m) = 4$ であるから、 $k = 2$ となる。

【0099】

ステップ(D)：プロセッサ30は、nとkとを引数としてsumサブルーチンを実行する。具体的には、プロセッサ30は、共有キャッシュメモリ20上にパケット50 (Pk2) を確保し、そのパケット50 (Pk2) の引数領域54に値n(=0)と値k(=2)とを格納する。次に、プロセッサ30は、sumの命令の先頭アドレスとパケット50 (Pk2) の先頭アドレスとをオペランドとして、fork命令を実行する。

【0100】

プロセッサ31とプロセッサ32はいずれも「空き状態」である。プロセッサ30は、優先度に従って「空き状態」のプロセッサ31の識別子を得る(図7のステップ(a)を参照)。プロセッサ30は、タスクの命令アドレスとタスクのパケットアドレスとをプロセッサ31に転送する(図7のステップ(b)を参照)。プロセッサ30は、パケット50 (Pk2) のロックビット領域51に"1"を書き込み、パケット50 (Pk2) のリターンビット領域52に"0"を書き込む(図7のステップ(d)を参照)。さらに、プロセッサ31は、パケット50 (Pk2) を参照しながらsumの命令の実行を開始する(図7のステップ(e)を参照)。このようにして、プロセッサ30とプロセッサ31とはsumサブルーチンを並列に実行する。

【0101】

ステップ(E)：プロセッサ30は、kとmとを引数としてsumサブルーチンを実行する。具体的には、プロセッサ30は、共有キャッシュメモリ20上にパケット50 (Pk3) を確保し、そのパケット50 (Pk3) の引数領域54に値k(=2)と値m(=4)とを格納する。次に、プロセッサ30は、sumの命令の先頭アドレスとパケット50 (Pk3) の先頭アドレスとをオペランドとして、exec命令を実行する。プロセッサ30がexec命令の実行を開始する前に、パケット50 (Pk1) はスタック領域に退避される。

【0102】

プロセッサ30は、パケット50 (Pk3) のロックビット領域51に"1"を書き込み、パケット50 (Pk3) のリターンビット領域52に"1"を書き込み、リターンアドレス領域53にexec命令の次の命令のアドレスを格納する(図7のステップ(f) 50



を参照)。また、プロセッサ30は、パケット50 (Pk3) を参照しながらsumの命令の実行を開始する(図7のステップ(g)を参照)。

【0103】

ステップ(F)：プロセッサ30は、ステップ(E)において呼び出したsumサブルーチンの実行を終了した後、スタック領域に退避したパケット50 (Pk1) を復帰させる。その後、プロセッサ30は、s1とs2とを加算する。ここで、s1は、ステップ(D)において実行されたsumサブルーチンの結果を示す。従って、s1は、パケット50 (Pk2) の戻り値領域55に格納される。s2は、ステップ(E)において実行されたsumサブルーチンの結果を示す。従って、s2は、パケット50 (Pk3) の戻り値領域55に格納される。プロセッサ30がステップ(E)において呼び出したsumサブルーチンの実行を終了した時点では、パケット50 (Pk2) を所有するタスクはまだ実行中である可能性がある。プロセッサ30は、パケット50 (Pk2) を所有するタスクの実行が終了した後、パケット50 (Pk2) の戻り値領域55に格納されている値を読み出し、その値をs1とする。ここでは、s1=3である。パケット50 (Pk2) を所有するタスクの実行が終了したか否かは、パケット50 (Pk2) のロックビット領域51の値を参照することにより判定される。パケット50 (Pk2) のロックビット領域51の値が"0"であることは、パケット50 (Pk2) を所有するタスクの実行が終了したことを示す。

【0104】

同様にして、プロセッサ30は、パケット50 (Pk3) を所有するタスクの実行が終了した後、パケット50 (Pk3) の戻り値領域55に格納されている値を読み出し、その値をs2とする。ここでは、s2=7である。プロセッサ30は、s1+s2を計算する。その結果、s=10が得られる。

【0105】

ステップ(H)：プロセッサ30は、sの値をパケット50 (Pk1) の戻り値領域55に格納する。その後、プロセッサ30は、unlock命令を実行する。

【0106】

プロセッサ30は、パケット50 (Pk1) のリターンビット領域52に格納されている値が"1"であるか否かを判定する(図8のステップ(h)を参照)。は、"1"である。従って、プロセッサ30は、パケット50 (Pk1) のロックビット領域51に"0"を格納し、リターンアドレス領域53に格納されているアドレスからの命令を実行する(図8のステップ(j)を参照)。この場合、mainのステップ(A)の次の命令から処理が再開される。

【0107】

ステップ(G)：ステップ(B)において、 $n+1=m$ であると判定された場合は、処理はステップ(G)に進む。プロセッサ30は、sに引数mの値を代入する。その後、処理はステップ(H)に進む。

【0108】

ここで、ステップ(D)において呼び出されたsumサブルーチンやステップ(E)において呼び出されたsumサブルーチンにおいても、上述したステップ(B)～(H)が実行されることに注意されたい。sumサブルーチンは、再帰呼び出し可能なサブルーチンだからである。

【0109】

このように、sumサブルーチンを再帰的に呼び出すことにより、1から4の和(1+2+3+4)を並列に計算することが達成される。この例では、ステップ(D)におけるfork命令とステップ(E)におけるexec命令によって2つのタスクが生成されている。fork命令は「空き状態」のプロセッサがある限りそのプロセッサにタスクを割り当てるために使用される命令であり、exec命令は、あくまで自プロセッサにタスクを割り当てるために使用される命令である。

【0110】

図11は、上述した処理の内容を模式的に示したものである。図11に示されるように、タスクsum(0, 4)からfork命令とexec命令とにより2つのタスクsum(0, 2)とタスクsum(2, 4)とが生成される。タスクsum(0, 2)はプロセッサ31に割り当てられ、タスクsum(2, 4)はプロセッサ30に割り当てられる。同様に、2つのタスクのそれぞれからさらに2つのタスクが生成される。「空き状態」のプロセッサが存在する限り他のプロセッサにタスクが割り当てられる。

【0111】

タスクsum(2, 4)からタスクsum(2, 3)とタスクsum(3, 4)とが生成される。しかし、いずれのタスクもプロセッサ30に割り当てられる。タスク(2, 3)の割り当て時に「空き状態」のプロセッサがすでに存在しなくなっているからである。 10

【0112】

このように、本発明のマルチプロセッサシステム1におけるプロセッサ30～32のそれぞれは、fork命令を解釈実行することにより、「空き状態」のプロセッサが存在する場合にはそのプロセッサにタスクを割り当て、「空き状態」のプロセッサが存在しない場合には実行中のタスクの実行を中断して、そのプロセッサにタスクを割り当てる。このようにして、処理すべきタスクが生成されると同時に「空き状態」のプロセッサか、あるいはタスクを生成したプロセッサのいずれかにその生成されたタスクが割り当てられる。その結果、生成したタスクは即時に実行される。これにより、従来のマルチプロセッサシステムでは必要とされた処理すべきタスクを保存する機構や、タスクの実行順序をスケジューリングする機構は不要となる。また、「空き状態」のプロセッサが存在する場合には 20

【0113】

さらに、fork命令やunlock命令は簡単なハードウェアで実現することができ、高速な処理も実現することができる。

【0114】

従って、集積回路上に実装されたマルチプロセッサシステム1において、例示した0から4までの和を求めるプログラムのような、タスクの処理時間がスケジューリング処理時間や実行待ちタスクの管理処理に要する時間に比べて小さいプログラムを並列処理する場合には、本発明のタスク実行方法は非常に有用である。

【0115】

なお、集積回路の外部から割り込みが入った場合には、プロセッサ状態管理装置22を用いて「空き状態」のプロセッサを検出し、「空き状態」のプロセッサのうち最も優先度の低いプロセッサに割り込み処理を行わせることにより、割り込み処理による性能低下を低減できる。

【0116】

なお、集積回路のプロセッサがすべて「空き状態」になったことは、プロセッサ状態管理装置22を用いて検出することができる。従って、この場合には、いずれかのプロセッサで例外処理を行うことによりデッドロックを回避することができる。

【0117】

以上のように、本発明によれば、あるプロセッサで新たなタスクを生成したときにそのタスクの実行を他あるいは自プロセッサによりただちに開始することができる。このことは、タスクを保持しておく機構やタスクの実行順序をスケジューリングする機構を不要にする。また、実行待ちのタスクを選択し、その選択されたタスクを「空き状態」のプロセッサに割り当てて処理も不要となる。 40

【0118】

その結果、タスクの処理時間に比較してプロセッサ割り当てに要する時間が少なくて済む。これにより、マルチプロセッサシステムにおいて、粒度の細かい並列処理の高速化を図ることができる。

【図面の簡単な説明】

【0119】

【図 1】 本発明のマルチプロセッサシステム 1 の構成を示す図である。

【図 2】 タスクの概念を模式的に示す図である。

【図 3】 マルチプロセッサシステム 1 におけるプロセッサ状態管理装置 2 2 の構成例を示す図である。

【図 4】 (a) および (b) は、プロセッサ状態管理装置 2 2 の動作の一例を説明する図である。

【図 5】 (a) および (b) は、プロセッサ状態管理装置 2 2 の動作の他の一例を説明する図である。

【図 6】 パケット 5 0 の構成を示す図である。

【図 7】 プロセッサ 3 0 ~ 3 2 が f o r k 命令を解釈実行する手順を示す図である。 10

【図 8】 プロセッサ 3 0 ~ 3 2 が u n l o c k 命令を解釈実行する手順を示す図である。

【図 9】 プロセッサの状態とタスクの状態とを説明する図である。

【図 1 0】 1 から 4 までの和を二分木に基づいて計算するプログラムの手順を示す図である。

【図 1 1】 図 1 0 に示すプログラムの処理の内容を模式的に示した図である。

【図 1 2】 従来のプロセッサ割当方法の動作を説明する図である。

【図 1 3】 タスクが中粒度～粗粒度である場合における、タスクの処理時間とオーバヘッドの処理時間とを示すタイムチャートである。

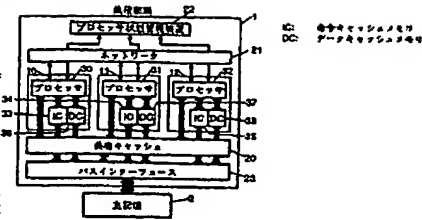
【図 1 4】 タスクが細粒度である場合における、タスクの処理時間とオーバヘッドの処理時間とを示すタイムチャートである。 20

【符号の説明】

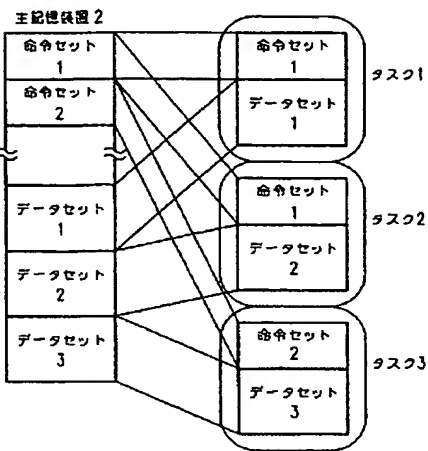
【 0 1 2 0 】

- 1    マルチプロセッサシステム
- 2    主記憶装置
- 1 0 ~ 1 2    要素プロセッサユニット
- 2 0    共有キャッシュ
- 2 1    ネットワーク
- 2 2    プロセッサ状態管理装置
- 2 3    バスインターフェース
- 3 0 ~ 3 2    プロセッサ
- 3 3 ~ 3 5    命令キャッシュ ( I C )
- 3 6 ~ 3 8    データキャッシュ ( D C )

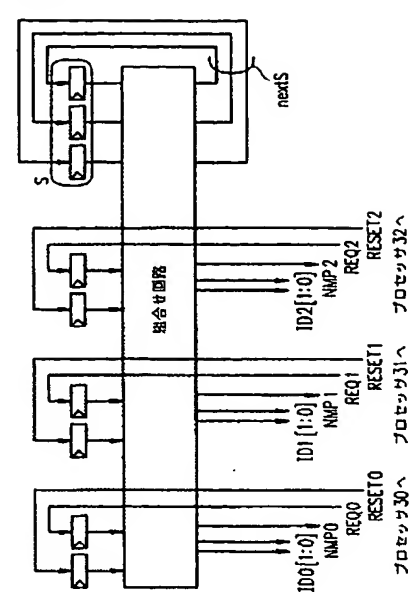
【図 1】



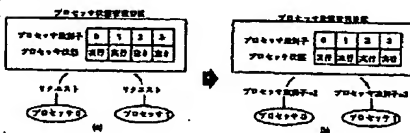
【図 2】



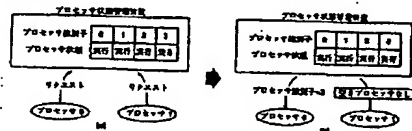
【図 3】



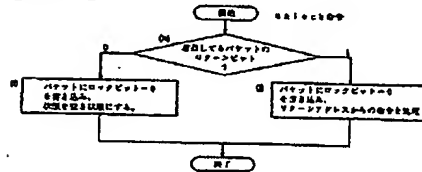
【図 4】



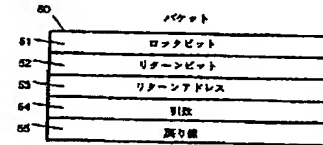
【図 5】



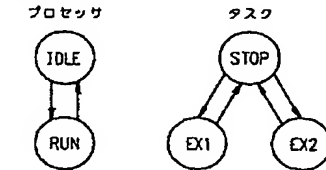
【図 8】



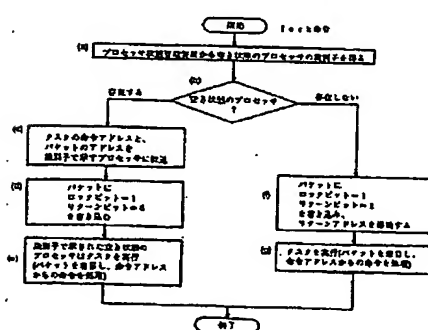
【図 6】



【図 9】

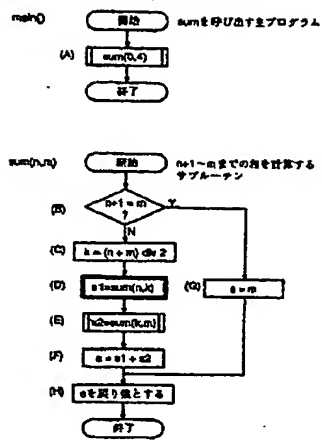


【図 7】

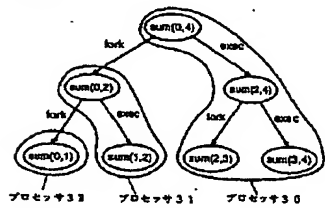


IDLE アイドル  
RUN 実行中  
STOP 実行待ちまたは実行済  
EX1 実行中(他タスク中断なし)  
EX2 実行中(他タスク中断あり)

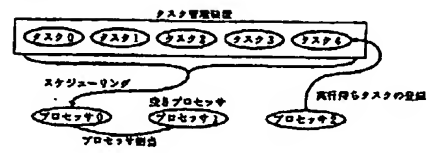
【圖 10】



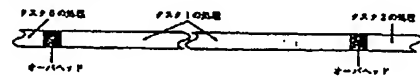
【图 1 1】



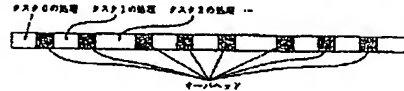
【圖 12】



【図 13】



【圖 14】



---

フロントページの続き

(72)発明者 田沼 仁

大阪府三島郡島本町広瀬4丁目11番6号

Fターム(参考) 58045 DD01 DD12 GG03 GG11

58098 AA10 GA04 GA07 GA08